Understanding XGBoost Algorithm In Detail

XGBoost or extreme gradient boosting is one of the well-known gradient boosting techniques(ensemble) having enhanced performance and speed in tree-based (sequential decision trees) machine learning algorithms. XGBoost was created by Tianqi Chen and initially maintained by the Distributed (Deep) Machine Learning Community (DMLC) group. It is the most common algorithm used for applied machine learning in competitions and has gained popularity through winning solutions in structured and tabular data. It is open-source software. Earlier only python and R packages were built for XGBoost but now it has extended to Java, Scala, Julia and other languages as well.

In this article, I'll be discussing how XGBoost works internally to make decision trees and deduce predictions.

To understand XGboost first, a clear understanding of decision trees and ensemble learning algorithms is needed.

Difference between different tree-based techniques:

XGBoost falls under the category of Boosting techniques in Ensemble Learning. Ensemble learning consists of a collection of predictors which are multiple models to provide better prediction accuracy. In Boosting technique the errors made by previous models are tried to be corrected by succeeding models by adding some weights to the models.

Basic Boosting Architecture:

Unlike other boosting algorithms where weights of misclassified branches are increased, in Gradient Boosted algorithms the loss function is optimised. XGBoost is an advanced implementation of gradient boosting along with some regularization factors.

Features of XGBoost:

- Can be run on both single and distributed systems(Hadoop, Spark).
- XGBoost is used in supervised learning(regression and classification problems).
- Supports parallel processing.
- Cache optimization.
- Efficient memory management for large datasets exceeding RAM.
- Has a variety of regularizations which helps in reducing overfitting.
- Auto tree pruning – Decision tree will not grow further after certain limits internally.
- Can handle missing values.
- Has inbuilt Cross-Validation.
- Takes care of outliers to some extent.

XGBoost Algorithm

Let's look at how XGboost works with an example. Here I'll try to predict a child's IQ based on age. For any basic assumption in such statistical data, we can take the average IQ and find how much variance(loss) is present. Residuals are the losses incurred will be calculated after each model predictions.

CHILD's AGE	CHILD's IQ	RESIDUALS
10	20	-10
15	34	4
16	38	8

So the average of 20, 34, and 38 is 30.67 for simplicity let's take it as 30. If we plot a graph keeping y-axis as IQ and x-axis as Age and then we can see the variance in points from the average mark.

At first, our base model(M0) will give a prediction 30. As from the graph, we know this model suffers a loss which will have some optimisation in the next model(M1). Model M1 will have input as age(independent features) and target as the loss suffered(variances) in M0. Until now it is the same as the gradient boosting technique.

For XGboost some new terms are introduced,

ƛ -> regularization parameter
ɣ -> for auto tree pruning
eta -> how much model will converge

Now calculate the similarity score,

Similarity Score(S.S.) = (S.R ^ 2) / (N + ƛ)
Here, S.R is the sum of residuals,
N is Number of Residuals

```
At first let's put  ƛ =0, then Similarity Score = (-10+4+8)^2 / 3+0 = 4/3 = 1.33
```

Let's make the decision tree using these residuals and similarity scores. I've set the tree splitting criteria as Age >10.

Again for these two leaves, we calculate the similarity scores which is 100 and 72.

After this, we calculate the

```
Gain = S.S of the branch before split - S.S of the branch after the split.
```

Gain = (100 + 72) – 1.3

Now we set our ɣ, which is a value provided to the model at starting and its used during splitting. If Gain>ɣ then split will happen otherwise not. Let's assume that ɣ for this problem is 130 then since the gain is greater than ɣ, further split will occur. By this method, auto tree pruning will be achieved. The greater the ɣ value more pruning will be done.

For regularization and preventing overfitting, we must increase the ƛ which was initially set to 0. But this should be done carefully as greater the ƛ value lesser the Similarity score, lesser the gain and more the pruning.

```
New prediction = Previous Prediction + Learning rate * Output
```

XGboost calls the learning rate as eta and its value is set to 0.3

For the 2nd reading(Age=15) new prediction = 30 + (0.3 * 6) = 31.8

The outcome is 6 is calculated from the average residuals 4 and 8.

New Residual = 34 – 31.8 = 2.2

Age	IQ	Residual
10	20	-7
15	34	2.2
16	38	6.2

This way model M1 will be trained and residuals will keep on decreasing, which means the loss will be optimized in further models.

Conclusion

XGboost has proven to be the most efficient Scalable Tree Boosting Method. It has shown outstanding results across different use cases such as motion detection, stock sales predictions, malware classification, customer behaviour

analysis and many more. The system runs way faster on a single machine than any other machine learning technique with efficient data and memory handling. The algorithm's optimization techniques improve performance and thereby provides speed using the least amount of resources.
